APPENDIX B

Garbage Collection

Garbage collection is a dynamic approach used for automatic memory management to reduce the memory leak problems. The garbage collection process identifies unused memory blocks and reallocates that storage for reuse. Garbage collection is implemented using the following approaches:

Mark-and-sweep In this approach when memory runs out, the garbage collection process locates all accessible memory and then reclaims the available memory.

Reference counting The garbage collection process here maintains a reference count of referencing number for each allocated object. When the memory count becomes zero, the object is marked as garbage and then destroyed by freeing its memory. The freed memory is finally returned to the memory heap.

Copy collection The garbage collection process maintains two memory partitions. When the first partition is full, the garbage collection process identifies all accessible data structures and copies them to the second partition. Then it compacts memory to allow continuous free memory.

Note Some programming languages like Java, C#, .NET, etc., have built-in garbage collection process to self-manage memory leak problem.

Advantages of Garbage Collection

Garbage collection frees the programmer from manually dealing with memory de-allocation, thereby eliminating or substantially reducing following types of programming bugs:

***Dangling pointer* bugs** are often encountered when the memory allocated to a variable (or an object) is freed but there are still pointers pointing to it. If such pointers are de-referenced especially when that memory is re-allocated to another variable or object, then results are simply unpredictable.

Double free bugs occur when the program tries to free a piece of memory that has already been freed. It may be a case that the freed memory has now been re-allocated to some other variable or object of the same or a different program. In such a case, the program will again give erroneous results.

Memory leaks occur when a program is unable to free memory occupied by objects that have become unreachable.

Garbage collection also helps in efficient implementation of persistent data structures.

Disadvantages of Garbage Collection

The typical disadvantages of garbage collection process include:

- Garbage collection consumes computing resources to decide which piece of memory must be freed. This information may already be available with the programmer.
- The time at which the garbage collection process will be executed is unpredictable which may lead to stalls scattered throughout a session. This is unacceptable in real-time environments, transaction processing, or in interactive programs. Although incremental, concurrent, and real-time garbage collectors solve this problem but with some or the other trade-off.
- Some of the bugs addressed by garbage collection can have certain security implications.

Requirements for Automatic Garbage Collection

An effective and efficient garbage collection process must have the following properties:

- Must identify garbage
- The object or variable identified as garbage must actually be garbage.
- Must have less overhead
- During garbage collection, the execution of the program is temporarily delayed. This delay must be minimum.